Proyecto Final Computación

Carla Domenech Cortina

Computación Evolutiva

Universidad San Jorge

Fecha: 29/05/2025

Profesor: Carlos Gracia Lázaro

INDICE

1.INTRODUCCIÓN

En este proyecto se ha trabajado en la optimización de hiperparámetros para un modelo de clasificación SVM (Support Vector Machine) utilizando algoritmos evolutivos. El objetivo principal es encontrar combinaciones de hiperparámetros que mejoren el rendimiento del modelo sobre un conjunto de datos determinado.

Para ello, se ha implementado un algoritmo evolutivo que genera una población inicial de posibles soluciones (combinaciones de hiperparámetros) y las evalúa mediante una función de fitness, que se basa en la precisión media obtenida mediante validación cruzada. Esta función permite medir la calidad de cada individuo y orientar la evolución hacia mejores soluciones.

Durante el proceso evolutivo, los individuos han sido seleccionados, cruzados combinando sus parámetros y mutados aleatoriamente para explorar nuevas regiones del espacio de soluciones.

Además, se ha incluido una comparación con el método de búsqueda aleatoria (random search) para analizar la eficacia del algoritmo evolutivo frente a una estrategia de selección no evolutiva. Se muestran también gráficos y estadísticas que permiten comparar el rendimiento de ambos métodos.

2.DESCRIPCIÓN DEL PROYECTO

En este proyecto se ha desarrollado un sistema de optimización de hiperparámetros utilizando un algoritmo evolutivo. El objetivo es mejorar el rendimiento de un modelo de clasificación basado en SVM (Support Vector Machine), un algoritmo de aprendizaje supervisado que encuentra el hiperplano óptimo para separar distintas clases.

Se han optimizado tres hiperparámetros clave del modelo SVM: C , gamma y el kernel (el cual es lineal o gaussiano). Como base de datos se ha empleado el conocido conjunto de datos Iris, que contiene muestras clasificadas en tres tipos de flores (iris setosa, iris versicolor e iris virginica). Cada una de estas flores se describe mediante cuatro características (longitud del sépalo, anchura del sépalo, longitud del pétalo y anchura del pétalo). El objetivo es ver la mejor combinación de hiperparámetros para que usando el modelo SVM sepamos distinguir que clase de flor es. Para ello, el sistema implementado utiliza operadores evolutivos como la generación aleatoria de individuos, selección por torneo binario, cruce entre padres y mutación, todo ello guiado por una función de fitness que mide el rendimiento del modelo. Además, se ha añadido una comparación con la técnica de búsqueda aleatoria (random search) para evaluar los resultados obtenidos por el algoritmo evolutivo. A continuación, se explicará en detalle el código desarrollado y las decisiones tomadas durante su implementación.

3.EXPLICACIÓN DEL CÓDIGO Y TOMA DE DECISIONES

Para poder implementar un algoritmo evolutivo con el objetivo de optimizar los hiperparámetros de un modelo SVM, se ha realizado un código que se ha estructurado en diferentes funciones.

Primero cargamos los datos de la base de datos iris y a continuación definimos el rango de los hiperparámetros que va a tener nuestro modelo. Se han definido rangos razonables para los hiperparámetros C y gamma, abarcando desde valores bajos hasta altos con el objetivo de explorar tanto modelos más simples como más complejos. Esto permite al algoritmo evaluar distintas configuraciones y ver cuál de ellas es la que mejor se ajusta a nuestro modelo. En cuanto al parámetro kernel, se ha limitado a dos opciones: lineal o rbf (también conocido como kernel gaussiano).

A continuación, se define una función llamada random_individual, la cual crea un individuo que contiene los tres hiperparámetros mencionados. Estos hiperparámetros se seleccionan de forma aleatoria dentro de los rangos definidos previamente.

Después, se ha creado la función evaluate (función de fitness), la cual construye un modelo SVM a partir de los hiperparámetros del individuo y calcula su precisión utilizando validación cruzada con 5 particiones. Elegimos 5 particiones porque es una configuración eficiente y suficiente para conjuntos de datos pequeños como el Iris. Si usaras más, el coste de entrenamiento crecería sin aportar una mejora significativa en la evaluación del modelo. A esta evaluación se le aplica una penalización por complejidad: si el kernel es de tipo rbf, se penaliza por ser más complejo, y cuanto mayor sea el valor de C, mayor será también la penalización. El hiperparámetro gamma no se penaliza porque está muy relacionado con el kernel rbf, el cual ya penalizamos, por lo que sería como penalizarlo dos veces. Finalmente, al valor de precisión obtenido se le resta esta penalización, y el resultado es el fitness que devuelve la función.

A continuación, se han definido los operadores evolutivos necesarios para hacer evolucionar la población. En primer lugar, se ha implementado un método de selección por torneo binario, en el que se escogen aleatoriamente dos individuos y se selecciona el que tenga mayor valor de fitness. Este método es sencillo y eficaz, ya que favorece a los mejores individuos sin eliminar del todo la diversidad, permitiendo que individuos no óptimos también tengan una pequeña probabilidad de ser seleccionados.

Para generar nuevos individuos, se ha definido una función de cruce que toma dos padres y construye un hijo combinando sus hiperparámetros. Para cada uno de los tres hiperparámetros (C, gamma y kernel), se elige aleatoriamente, con una probabilidad del 50%, si se hereda del primer o del segundo padre. De este modo, el nuevo individuo puede estar formado íntegramente por los parámetros de uno de los padres o ser una combinación de ambos, lo que favorece la diversidad genética dentro de la población y permite explorar nuevas combinaciones prometedoras a partir de soluciones ya buenas.

Además, se ha implementado una función de mutación que introduce variaciones aleatorias sobre un individuo ya existente. Esta función modifica los valores de los hiperparámetros C y gamma con una cierta probabilidad, aplicando un cambio porcentual

aleatorio dentro de un rango. También se permite, aunque con menor probabilidad, cambiar el tipo de kernel. Se ha utilizado una tasa de mutación alta (0.8) en C y gamma para fomentar la exploración, ya que son valores continuos, mientras que en kernel se ha usado una tasa más baja (0.3) al ser un parámetro discreto que cambia el modelo de forma más drástica. El cambio que se aplica a los valores de C y gamma es de +- 70%, es decir, multiplicar el valor de C por un numero entre 0.3 y 1.7, así permite variaciones amplias pero controladas, explorando nuevas configuraciones sin alterar completamente el valor original.

Gracias a la combinación de estos tres operadores: selección, cruce y mutación, el algoritmo es capaz de mejorar progresivamente la calidad de los individuos generación tras generación.

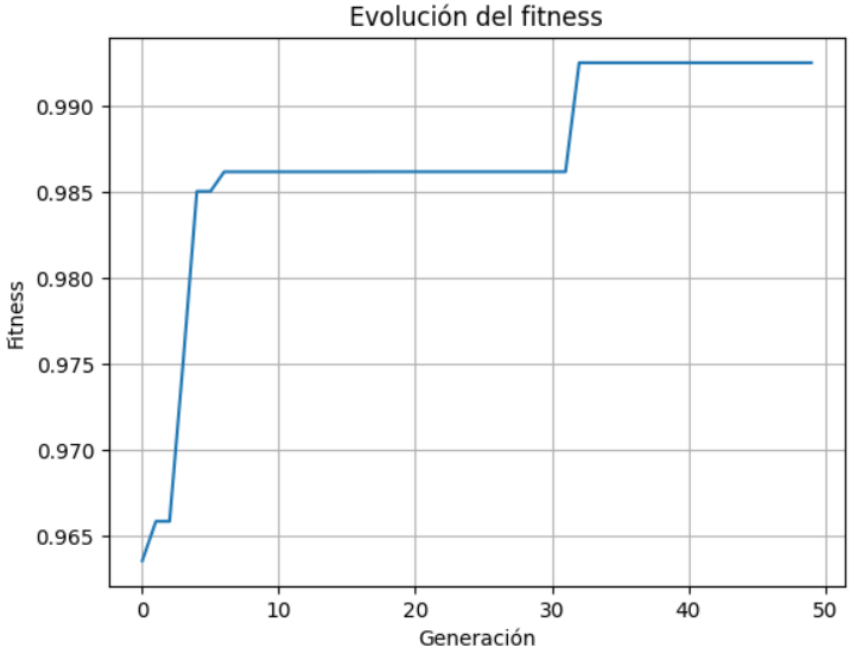
Por último, se ha implementado la función principal del algoritmo evolutivo, que gestiona todo el proceso generación tras generación. Se comienza creando una población inicial de individuos generados aleatoriamente. En cada generación, todos los individuos son evaluados mediante la función de fitness, y se selecciona al mejor de ellos para ser conservado (elitismo). A continuación, para el resto de la población, se generan nuevos individuos mediante selección por torneo binario, cruce y mutación, hasta completar la nueva población. Este proceso se repite durante un número determinado de generaciones, y en cada una de ellas se almacena el mejor valor de fitness alcanzado. Se ha optado por un tamaño de población de 20 individuos y un total de 50 generaciones, ya que esta configuración ofrece un buen equilibrio entre diversidad de soluciones y tiempo de ejecución, siendo suficiente para observar una mejora progresiva en el rendimiento del modelo. Más adelante se analizarán los resultados obtenidos.

Para realizar la parte extra del trabajo, se ha implementado un enfoque de búsqueda aleatoria (random search) con el fin de comparar su rendimiento con el del algoritmo evolutivo. Random search es una técnica no evolutiva que consiste en generar combinaciones aleatorias de hiperparámetros y evaluar directamente su rendimiento, sin aplicar operadores como selección, cruce o mutación. Aunque no aprovecha información de generaciones anteriores ni mejora progresivamente los individuos, puede ser útil como método de referencia.

En el código implementado, se realiza un número determinado de intentos (n_trials) en los que se generan individuos aleatorios, se evalúan con la misma función de fitness y se registran tanto sus valores de fitness como el mejor valor acumulado hasta cada intento. Al finalizar, se identifica el mejor individuo encontrado y se devuelve junto con el historial de resultados. En la sección de análisis de resultados se compararán ambos enfoques, evaluando cuál de los dos ofrece mejores soluciones para el problema planteado.

4.ANÁLISIS DE LOS RESULTADOS

En primer lugar, se analizan los resultados obtenidos tras ejecutar el algoritmo evolutivo durante 50 generaciones con una población de 20 individuos. Esta configuración se eligió para mantener un equilibrio entre diversidad y coste computacional. Como se observa en la gráfica de evolución del fitness, el valor del fitness mejora de forma progresiva durante las primeras generaciones, especialmente entre la generación 1 y la 7, alcanzando una mejora significativa. Posteriormente, el algoritmo entra en una fase constante donde apenas hay variación en el fitness, lo cual indica que se ha alcanzado un fitness bueno y se mantiene estable. Sin embargo, alrededor de la generación 33 se produce una mejora puntual que eleva el fitness a su valor máximo (0.9925), manteniéndose así hasta la generación final. Este comportamiento es típico en los algoritmos evolutivos, donde la fase inicial de exploración permite descubrir soluciones prometedoras y, posteriormente, la explotación se centra en mejorar esas buenas soluciones encontradas.



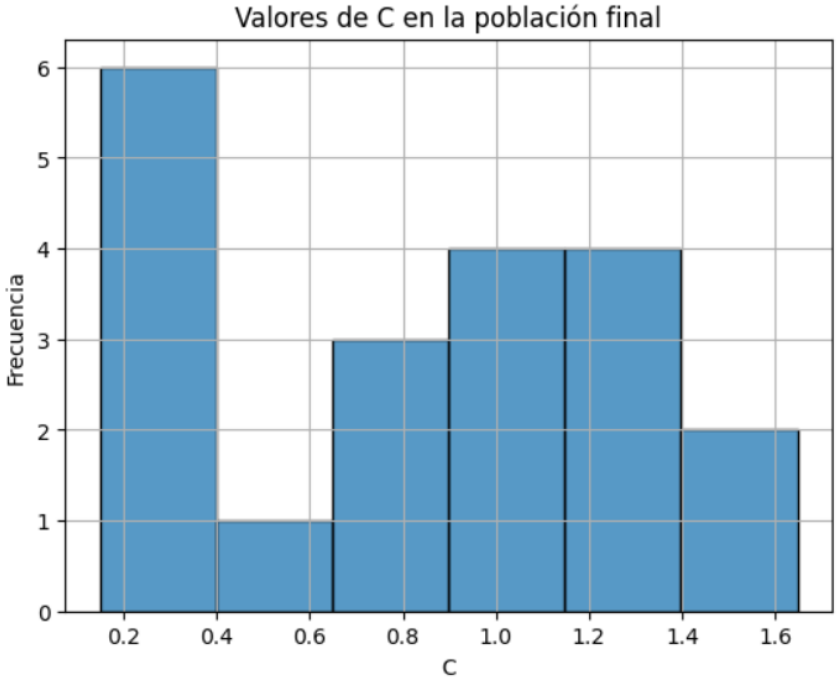
En cuanto a los valores de los hiperparámetros presentes en la población final, se observa que el parámetro C se concentra en su mayoría en valores bajos, especialmente por debajo de 1.5. Esto sugiere que el modelo ha favorecido soluciones con menor complejidad (menos penalización), lo que coincide con los buenos resultados obtenidos en términos de fitness.

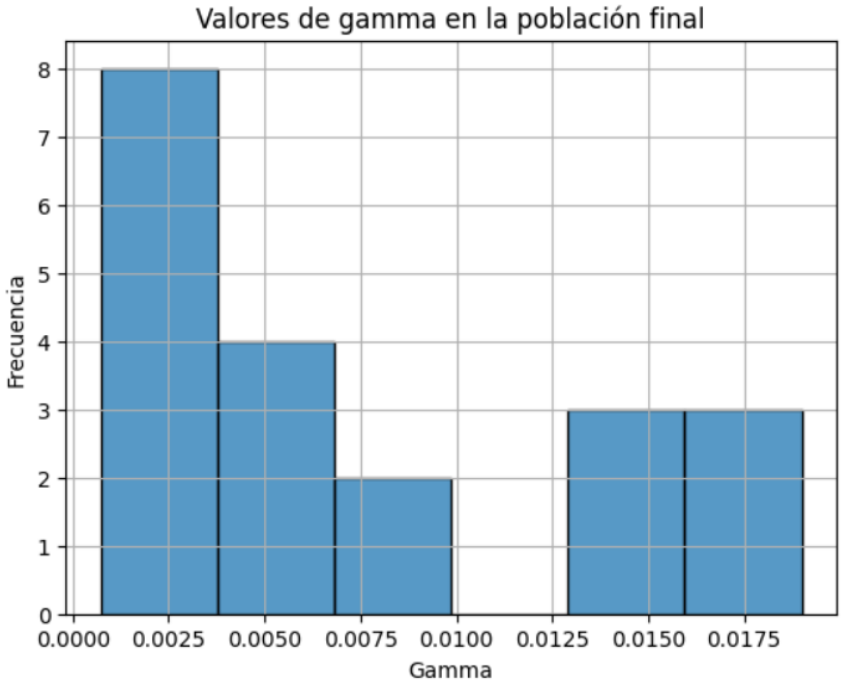
Por su parte, los valores de gamma también tienden a situarse en rangos bajos, siendo especialmente frecuentes aquellos cercanos a 0.002. Esto refuerza la idea de que configuraciones menos complejas (valores bajos de C y gamma) han proporcionado modelos más generalizables para este problema.

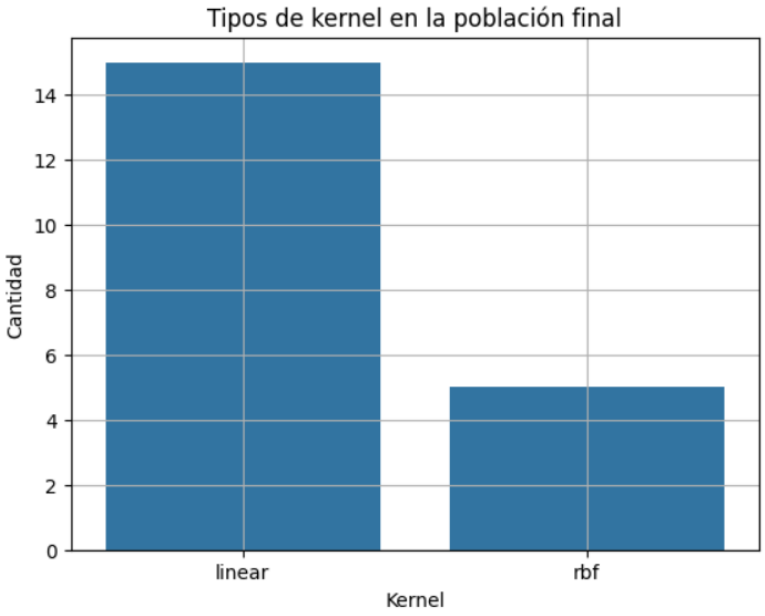
Respecto al tipo de kernel, se observa un claro predominio del kernel lineal frente al kernel rbf. Este resultado está relacionado con la penalización aplicada a modelos con kernel rbf

y también indica que, para el conjunto de datos Iris, un modelo lineal es suficiente para obtener una buena separación entre las clases.

En resumen, los resultados muestran que el algoritmo evolutivo ha sido capaz de encontrar configuraciones de hiperparámetros eficaces, priorizando modelos simples y bien ajustados, lo que ha permitido alcanzar un buen fitness.







Una vez evaluado el algoritmo evolutivo, se ha ejecutado el algoritmo de búsqueda aleatoria (random search) con el objetivo de comparar su rendimiento. En este caso, se realizaron 50 pruebas (n_trials), el mismo número de iteraciones que generaciones tiene el algoritmo evolutivo, para poder compararlos. En cada intento, se generó un conjunto aleatorio de hiperparámetros, se evaluó su rendimiento y se registró el mejor fitness acumulado hasta ese punto.

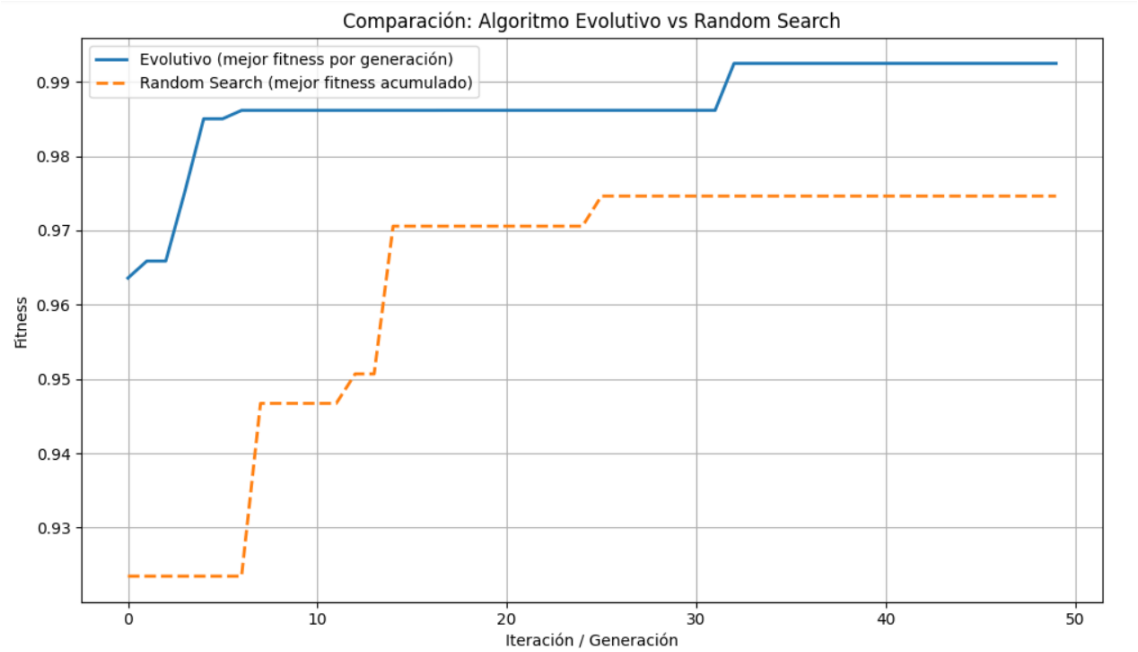
Tal como se observa en los resultados, el mejor individuo encontrado por random search obtuvo un fitness de 0.9746. Por otro lado, el mejor individuo encontrado por el algoritmo evolutivo alcanzó un fitness de 0.9925. Además, si comparamos las medias de fitness obtenidas a lo largo de todas las generaciones o intentos, el algoritmo evolutivo logró una media de 0.9869 frente a 0.9080 del random search.

Esta diferencia también se refleja en la gráfica de comparación, donde se puede observar que la curva del algoritmo evolutivo alcanza valores más altos de fitness desde el principio y los mantiene, mientras que la curva de random search progresa de forma más lenta y alcanza un límite inferior. Esto es lógico, ya que el algoritmo evolutivo mejora iterativamente su población mediante selección, cruce y mutación, aprovechando la información obtenida en generaciones anteriores, mientras que random search no aplica ninguna estrategia de mejora y depende únicamente del azar.

En resumen, el algoritmo evolutivo ha demostrado ser más eficaz a la hora de encontrar soluciones óptimas en este problema de optimización de hiperparámetros para SVM, obteniendo mejores resultados tanto en términos de fitness máximo como de promedio. Random search, aunque más simple, ha servido como referencia para mostrar las ventajas del enfoque evolutivo.

```
Mejor individuo encontrado por Random Search:
{'C': 5.371247646936878, 'gamma': 3.3066198842083363, 'kernel': 'linear'}
Fitness: 0.9746

Comparativa:
Media Fitness Evolutivo: 0.9869
Media Fitness Random Search: 0.9080
```



5.CONCLUSIÓN

En este proyecto se ha desarrollado e implementado un algoritmo evolutivo con el objetivo de optimizar los hiperparámetros de un modelo SVM aplicado al conjunto de datos Iris. A través de distintas funciones diseñadas para evaluar individuos, así como para realizar operaciones evolutivas como la selección, el cruce y la mutación, se ha logrado construir un sistema capaz de mejorar progresivamente la precisión del modelo. Además, se ha incorporado una comparación con el enfoque de búsqueda aleatoria (random search), lo que ha permitido demostrar las ventajas del enfoque evolutivo.

Los resultados muestran que el algoritmo evolutivo alcanza un mayor fitness en comparación con el random search. Esto confirma la utilidad de los algoritmos evolutivos en tareas de optimización de hiperparámetros y resalta su capacidad para encontrar soluciones eficientes. En conjunto, el proyecto ha permitido aplicar de manera práctica conceptos de computación evolutiva y aprendizaje automático, mostrando su efectividad.